Exercise Sheet 6

Exercise 1 (Scheduling Strategies)

1. Why exists a system idle process in some operating systems?

2. Explain the difference between preemptive and non-preemptive scheduling.

3. Name one drawback of preemptive scheduling.

4. Name one drawback of non-preemptive scheduling.

5. How does multilevel feedback scheduling work?

6. Which scheduling strategies are fair?
   A scheduling method is fair *when each process gets the CPU assigned at some point.*

   ☐ Priority-driven scheduling
   ☐ First Come First Served
   ☐ Last Come First Served
   ☐ Shortest Job First
   ☐ Longest Job First
   ☐ Shortest Remaining Time First
   ☐ Longest Remaining Time First
   ☐ Round Robin with time quantum
   ☐ Highest Response Ratio Next
   ☐ Earliest Deadline First

7. Which scheduling strategies can operate preemptive?
   - ☐ First Come First Served
   - ☐ Shortest Job First
   - ☐ Longest Job First
   - ☐ Shortest Remaining Time First
   - ☐ Longest Remaining Time First
   - ☐ Round Robin with time quantum
   - ☐ Static multilevel scheduling
   - ☐ Multilevel feedback scheduling

8. Which scheduling strategies require an estimation of the CPU runtime (= execution time)?
   - ☐ Priority-driven scheduling
   - ☐ First Come First Served
   - ☐ Last Come First Served
   - ☐ Shortest Job First
   - ☐ Longest Job First
   - ☐ Shortest Remaining Time First
   - ☐ Longest Remaining Time First
   - ☐ Round Robin with time quantum
   - ☐ Highest Response Ratio Next
   - ☐ Earliest Deadline First

Exercise 2 (Scheduling)

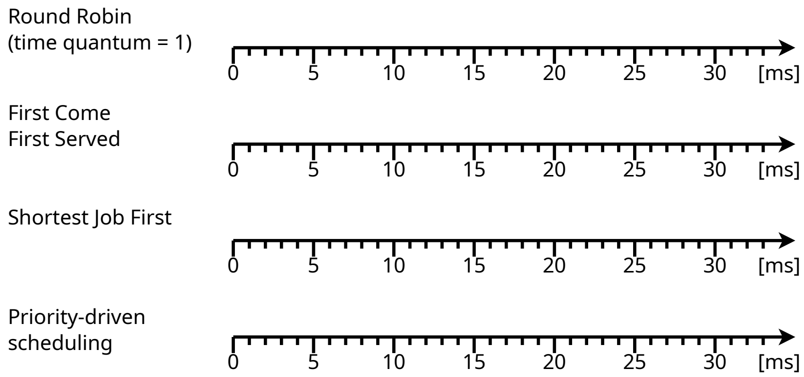
Process	**CPU time**	**Priority**
A	5 ms	4
B	10 ms	15
C	3 ms	12
D	6 ms	5
E	8 ms	7

Five processes shall be processed on a single CPU/core system. All processes are at time point 0 in state `ready` and created in the given order. High priorities are characterized by smaller values.

Draw the execution order of the processes with a Gantt chart (timeline) for **Round Robin** (time quantum $q = 1\,\text{ms}$), **FCFS**, **SJF**, and **priority-driven scheduling**.

The Priority column in the table is only relevant for the priority-driven scheduling and not for Round Robin or FCFS.

Calculate the average runtimes and average waiting times of the processes.



The CPU time is the time that the process needs to access the CPU to complete its execution. Runtime = *lifetime* = time period between the creation and the termination of a process = (CPU time + waiting time).

Runtime	**A**	**B**	**C**	**D**	**E**
RR					
FCFS					
SJF					
Priority-driven scheduling					

Waiting time = time of a process being in state `ready`. Waiting time = runtime - CPU time.

Waiting time	**A**	**B**	**C**	**D**	**E**
RR					
FCFS					
SJF					
Priority-driven scheduling					

Exercise 3 (Inter-Process Communication)

1. Describe what a critical section is.

2. Describe what a race condition is.

3. Describe how to avoid race conditions.

Exercise 4 (Communication of Processes)

1. What must be considered, when inter-process communication via shared memory segments is used?

2. What is the function of the shared memory table in the Linux kernel?

3. What is the impact of a restart (reboot) of the operating system on the existing shared memory segments?
   (Only a single answer is correct!)
   ☐ The shared memory segments are created new during boot and the contents are restored.
   ☐ The shared memory segments are created new during boot, but they remain empty. This means, only the contents are lost.
   ☐ The shared memory segments and their contents are lost.
   ☐ Only the shared memory segments are lost. The operating system stores the contents in temporary files inside the folder `\tmp`.

4. According to which principle operate message queues?
   (Only a single answer is correct!)
   ☐ Round Robin ☐ LIFO ☐ FIFO ☐ SJF ☐ LJF

5. How many processes can communicate with each other via a pipe?

6. What is the effect, when a process tries to write data into a pipe without free capacity?

7. What is the effect, when a process tries to read data from an empty pipe?

8. Which two different types of pipes exist?

9. Which two different types of sockets exist?

10. Communication via pipes works...
    - ☐ memory-based
    - ☐ stream-based
    - ☐ object-based
    - ☐ message-based

11. Communication via message queues works...
    - ☐ memory-based
    - ☐ stream-based
    - ☐ object-based
    - ☐ message-based

12. Communication via shared memory segments works...
    - ☐ memory-based
    - ☐ stream-based
    - ☐ object-based
    - ☐ message-based

13. Communication via sockets works...
    - ☐ memory-based
    - ☐ stream-based
    - ☐ object-based
    - ☐ message-based

14. Which types of inter-process communication operate bidirectional?
    - ☐ Shared memory segments
    - ☐ Message queues
    - ☐ Anonymous pipes
    - ☐ Named pipes
    - ☐ Sockets

15. Name a sort of inter-process communication, which can only be used for processes, which are closely related to each other.
    - ☐ Shared memory segments
    - ☐ Message queues
    - ☐ Anonymous pipes
    - ☐ Named pipes
    - ☐ Sockets

16. Which sort of inter-process communication allows communication over computer boundaries?
    - ☐ Shared memory segments
    - ☐ Message queues
    - ☐ Anonymous pipes
    - ☐ Named pipes
    - ☐ Sockets

17. With which sorts of inter-process communication remains the data intact without a bound process?
    - ☐ Shared memory segments
    - ☐ Message queues
    - ☐ Anonymous pipes
    - ☐ Named pipes
    - ☐ Sockets

18. For which sort of inter-process communication guarantees the operating system <u>not</u> the synchronization?
    - ☐ Shared memory segments
    - ☐ Message queues
    - ☐ Anonymous pipes
    - ☐ Named pipes
    - ☐ Sockets

Exercise 5 (Synchronization)

1. What is the advantage of signal and wait compared with busy waiting?

2. Which two problems can arise from locking?

3. What is the difference between signaling and locking?

4. Which four conditions must be fulfilled at the same time as precondition that a deadlock can arise?

☐ Recursive function calls	☐ Hold and wait
☐ Mutual exclusion	☐ > 128 processes in `blocked` state
☐ Frequent function calls	☐ Iterative programming
☐ Nested `for` loops	☐ Circular wait
☐ No preemption	☐ Queues

5. Does a deadlock occur? Perform the deadlock detection with matrices.

$$\text{Existing resource vector} = \begin{pmatrix} 8 & 6 & 7 & 5 \end{pmatrix}$$

$$\text{Current allocation matrix} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 0 & 4 \\ 0 & 2 & 1 & 1 \end{bmatrix} \qquad \text{Request matrix} = \begin{bmatrix} 3 & 2 & 4 & 5 \\ 1 & 1 & 2 & 0 \\ 4 & 3 & 5 & 4 \end{bmatrix}$$

Exercise 6 (Cooperation of Processes)

1. What is a semaphore and what is its intended purpose?

2. Which two operations are used with semaphores?

3. What is the difference between semaphores versus locks?

4. What is a binary semaphore?

5. What is a mutex and what is its intended purpose?

6. Which type of semaphores has the same functionality as the mutex?

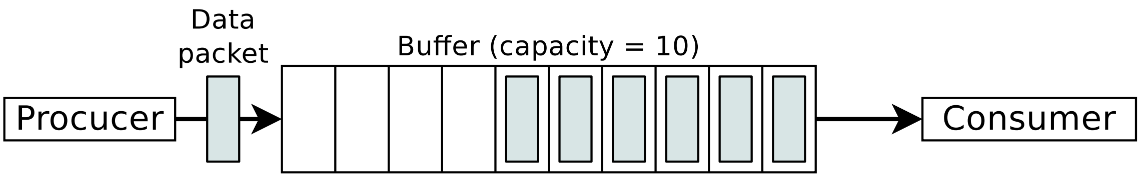
7. Which states can a mutex have?

8. Which Linux/UNIX command returns information about existing shared memory segments, message queues and semaphores?

9. Which Linux/UNIX command allows to erase existing shared memory segments, message queues and semaphores?

Exercise 7 (Producer/Consumer Scenario)

A producer should send data to a consumer. A buffer with limited capacity should be used to minimize the waiting times of the consumer. Data is placed into the buffer by the producer and the consumer removes data from the buffer. Mutual exclusion is necessary in order to avoid inconsistencies. If the buffer has no more free capacity, the producer must block itself. If the buffer is empty, the consumer must block itself.



For synchronizing the two processes, create the required semaphores, assign them initial values and insert semaphore operations.

```
typedef int semaphore;          // semaphores are of type integer
```

```

void producer (void) {
    int data;

    while (TRUE) {                // infinite loop
        createDatapacket(data);    // create data packet
```

```
        // write data packet into the buffer
        insertDatapacket(data);
```

```


    }
}

void consumer (void) {
    int data;

    while (TRUE) {                // infinite loop
```

```
        // pick data packet from the buffer
        removeDatapacket(data);
```

```
        // consume data packet
        consumeDatapacket(data);
    }
}
```

Exercise 8 (Semaphores)

In a warehouse, packages are delivered constantly by a supplier and picked up by two deliverers. The supplier and the deliverers need to pass through a gate. The gate can always be passed only by a single person. The supplier brings three packages with every shipment to the incoming goods section. One of the deliverers can pick two packages with every pickup from the outgoing goods section. The other deliverer can pick only a single package per pickup from the outgoing goods section.

Exactly one process `Supplier`, one process `Deliverer_X` and one process `Deliverer_Y` exist. For synchronizing the three processes, create the required semaphores, assign them values and insert semaphore operations. These conditions must be met:

- Only a single process can pass through the gate.
  It is impossible that multiple processes pass though the gate simultaneously.
- Only one of both existing deliverers can access the outgoing goods section.
  It is impossible that both deliverers access the outgoing goods section simultaneously.
- It should be possible that the supplier and one of the deliverers can simultaneously unload and pick goods.
- The capacity of the warehouse is 10 packages.
- No deadlocks are allowed.
- At the beginning, the warehouse contains no packets and the gate, as well as the incoming goods section and the outgoing goods section are free.

Source: TU-München, Übungen zur Einführung in die Informatik III, WS01/02

```
Supplier
{
  while (TRUE)
  {

    <Pass through gate>;

    <Enter incoming
    goods section>;

    <Unload 3 packets>;

    <Leave incoming
    goods section>;

    <Pass through gate>;
```

```
Deliverer_X
{
  while (TRUE)
  {

    <Pass through gate>;

    <Enter outgoing
    goods section>;

    <Pick 2 packets>;

    <Leave outgoing
    goods section>;

    <Pass through gate>;
```

```
Deliverer_Y
{
  while (TRUE)
  {

    <Pass through gate>;

    <Enter outgoing
    goods section>;

    <Pick 1 packet>;

    <Leave outgoing
    goods section>;

    <Pass through gate>;
```